5/14/2020

Aleksey Garbaly

Rachel Rystedt

Amit Bista

Making BERT Better

Project Motivation

The teams project motivation was to improve an existing sentiment analysis models using a state-of-the-art (SOTA) model such as BERT. Sentiment analysis models offer much added value to businesses, by understanding your customers emotions toward a product or service you are able to better serve them and improve upon your products or services. This needs to automate sentiment becomes more apparent in a world in which people offer more and more opinions online. BERT and other NLP models are often large and costly to train, limiting the amount of people that could use them and improve upon them. In this paper we explore various ways to make BERT better to make it more available to others. When optimal parameters are found, they could be combined to get the best possible BERT model.

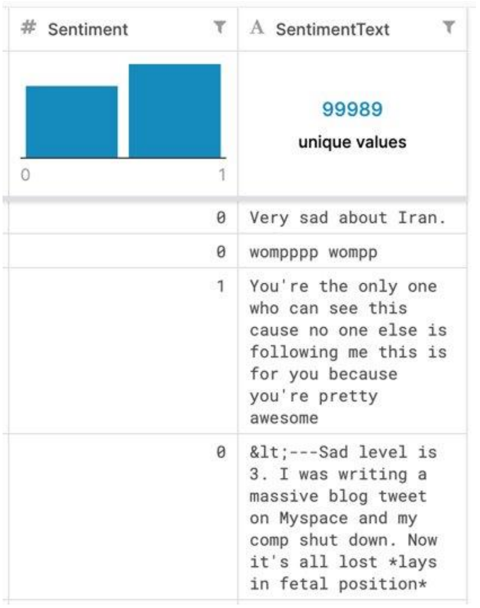
Data

Data set

The dataset the team choose came from a text processing challenge on Kaggle, the URL for the dataset was found on: https://www.kaggle.com/c/twitter-sentiment-analysis2

The testing set for the data had 299,989 rows with two columns, the ItemID and the text of the tweet. The training set contained 99,989 rows of strings, with ItemID, Sentiment and the text of the tweet as seen in Figure 1.

In total we had around 400,000 rows of strings to work with. Our data was evenly distributed with, positive sentiment (1) tweets having slightly more instances, but the team did not feel that the dataset was too skewed in one direction. The testing dataset was not used because it had no sentiment labels so it could only be used for prediction but not for model evaluation.



Sentiment	SentimentText
0 1	99989 unique values
0	Very sad about Iran.
0	wompppp wompp
1	You're the only one who can see this cause no one else is following me this is for you because you're pretty awesome
0	&lt;---Sad level is 3. I was writing a massive blog tweet on Myspace and my comp shut down. Now it's all lost *lays in fetal position*

Figure 1: Kaggle dataset summary

Data Preparation Steps

The team went through data preparation steps to ensure the data was ready to be input into BERT. First regular expressions were used to clean any strings that the team felt were not deterministic for sentiment analysis. An example of this is removing words that began with @, this symbol doesn't help our model predict if a tweet was positive or negative as it seemed as if it had replaced the hashtag [1]. Removing a hashtag may influence sentiment but we wanted to extract sentiment from sentences, not being influenced by the subject the sentence was directed toward. A custom package [2] for sentiment analysis was used to remove stop words in the tweets, removing additional noise. We also removed numbers as they were additional data in the tweets that did not help predict a tweets sentiment. We also removed blank tweets as they could not be tokenized by the model and caused problems when training.

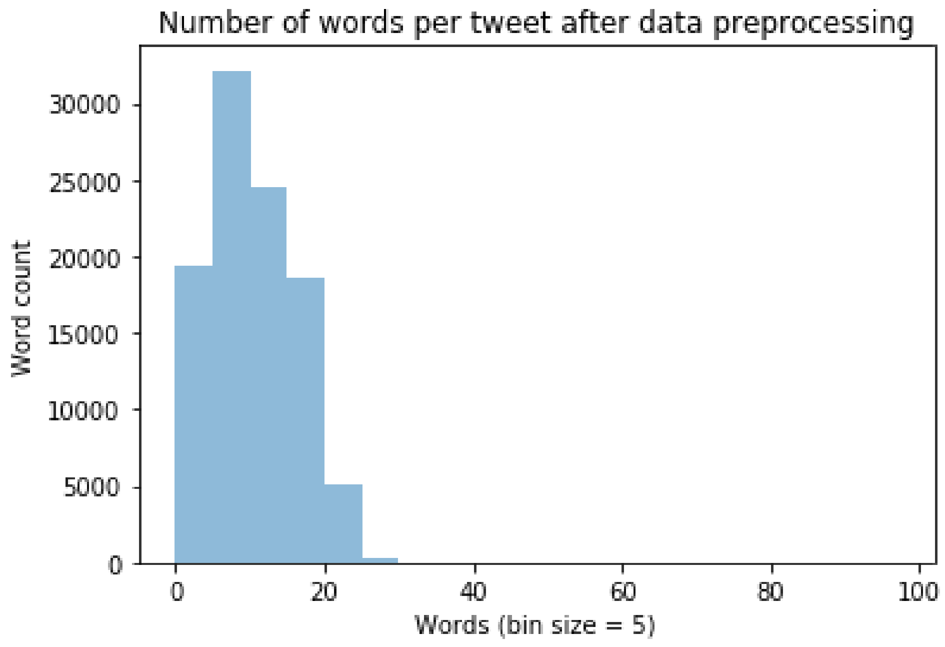


Figure 2: histogram showing distribution of words per tweet

After the data preprocessing a histogram was produced as seen I Figure 2 which displayed the number of words per tweet, this number helped determine the sequence length used in our model.

Tools & Methods

Models and Environments

Google Colab

Setting up local environments became a time-consuming process for the team, as we ran into installation issues with Transformers, a needed package for BERT. To solve for this the team used Google Colab as the cloud server to execute our code on. The model used in the paper was based on a prewritten Google Colab notebook found in the citation [3].

BERT

BERT, Bidirectional Encoder Representations from Transformers, was introduced by researchers at Google AI Language [4]. It provides state-of-the-art results in various NLP tasks that includes Question Answering (SQuADv1.1), Natural Language Inference (MNLI) etc. It applies the bidirectional training of Transformer. Transformer is an attention model in the language modelling world as it learns the contextual relations between words in a text. It has two separate mechanisms - encoder and decoder. An encoder reads the text input and a decoder predicts for the task. Using bidirectionally trained model, BERT has deeper sense of language context and flow provided by Transformer than that of single-direction language models. Instead of reading the text input sequentially, the Transformer encoder reads the entire sequence of words at once. Therefore, it is considered bidirectional, but it would be more accurate to consider BERT as non-directional.

- BERT is available in two different sizers - BERT BASE and BERT LARGE.
- It has 24 Transformer blocks, 1024 hidden layers, and 340 million parameters.

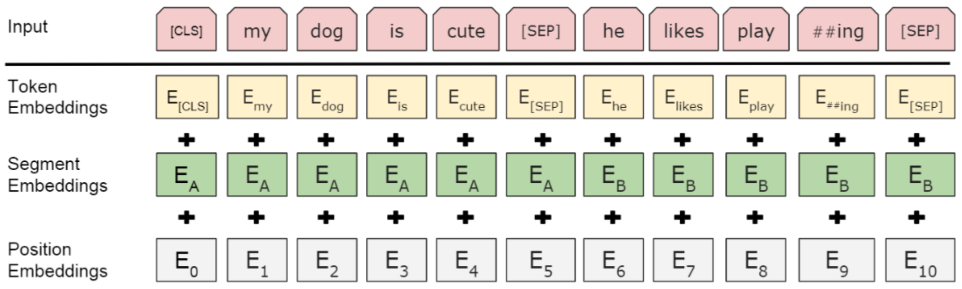


Figure 3: BERT input representation, visualizing how words are tokenized

- It is pre-trained on 40 epochs over a 3.3-billion-word corpus that includes 800 million words from Books Corpus and 2.5 billion words from English Wikipedia.
- The pre-training runs on 16 TPUs for training.
- It takes a sequence of words as an input that flows up the stack where each layer applies self-attention and the results are passed to the next layer through a feed-forward network and then the next encoder.
- The dimensionality or the output of each encoder position is a vector of size, which is called hidden_size. The BERT Base has 768.

F1 Score

While there are many metrics that can measure how well models perform for any given tasks, the f1 score is a balanced one because it provides a combination precision and recall. Precision is a measure of how exact your predictions are while the recall compares how many predicted true positives there are compared to the actual number of positives, it can be thought as the sensitivity. Equation (1) shows how the f1 score is calculated.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \qquad (1)$$

Implemented Model Improvements

MobileBERT

MobileBERT was used as a way for our team to quickly test out if we could get hyperparameter tuning to improve the model's accuracy [5]. MobileBERT is a lite version of BERT Base that has bottleneck layers to help remove noise and increase processing power as shown ion Figure 4. The paper for MobileBERT show that it is 4.3x smaller than BERT base and 5.5x faster.[1]

[1] https://arxiv.org/pdf/2004.02984.pdf

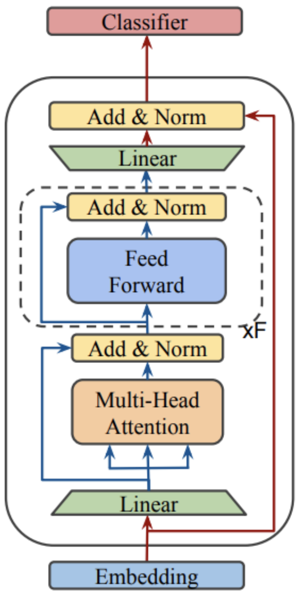


Figure 4: Architecture of MobileBERT

The model was trained using various epochs (1 to 5) but the team did not find significant improvement in the model. The f1_scores and the loss function comparison for different epochs help to draw a conclusion that the model provides the result that the team is looking for even with only 1 epoch. Using 1 epoch as a base, the team also experimented on various Learning Rates and Batch Sizes, and the results did not have significant changes in f1_score and loss function.

1 Epoch is Enough

This method for BERT improvement came from a paper that looked at trying to reduce training time and improve the overall model [6]. They found that by training on a larger dataset and training on less epochs, they sped up the training and reduced overfitting without needing to perform regularization as it slowed down the training. When they replaced 10 epochs with one, they saw a 1.9-3.3x reduction in training time and when they trained with different data sizer and iteration parameters, they were able to reduce training time by 1-2.7x. Generally, they state the following "Training for E epochs is roughly equivalent to training on a shuffled dataset consisting of E copies of the original dataset for one epoch" (One Epoch Is All You Need, 2019).

Failed Model Improvements

ALBERT

Initially we wanted to use the best SOTA model that involved BERT, which was ALBERT [7]. Unfortunately, due to its recent development there were few readily available models that were made easy for students such as us. While we did find one, we had problems with setting up and problems with preparing data for it, so we decided to use its predecessor, BERT.

Are Sixteen Heads Really Better than One

There was a paper [8] that looked at models such as BERT that had multiple heads looking at different parts of the input. The more heads a model has the more parts it can direct its attention to at the same time during training. It was found that that by removing 20% - 40% of the heads, the training speed could be decreased with a negligible impact on performance. The model became more efficient after pruning the heads, the higher batch size was used, the better the efficiency. This method was not implemented because we were not able to easy change the architecture of BERT using out notebook and doing so would require a lot of time.

Lamb Optimizer

In early 2020 a paper was published that looked at speeding up BERT by using a different optimizer that was meant to run on Google's TPUv3's [9]. The LAMB optimizer uses a layer-wise adaption strategy that increased the batch size to match the TPUv3's memory limit, reducing the training time from 3 days to 76 minutes. While Google Colab does not give public access to TPUv3, they do allow using TPUv2 which have larger memory limits that other GPU's on the market. This optimizer was not utilized because the notebook had no place to easily change BERT's default optimizer, Adam.

Switching Optimizers

It is well known that adaptive optimizers like Adam are generally fast but sometimes tend to overshoot while stochastic optimizers such as SGD are slower but tend to be more accurate. A paper was written [10] that looked at switching optimizers during model training to take advantage of the speed of Adam ad the accuracy of SGD. They developed SWAT (Switch from Adam to SGD), which switched optimizers when a certain gradient condition was met. After running the method on many benchmarks, it was found that the results were comparable to SGD while keeping properties from Adam such as rapid initial progress and hyperparameter insensitivity. As with the previous method, we were unable to manipulate the default optimizer in our notebook environment.

DistilBERT

BERT is a model that takes a long time to train and since it came out there has been many proposed variants of BERT that are faster, smaller, or more accurate. DistilBERT is a variant of BERT that reduces its size by 40% while being 60% faster and retaining 97% accuracy [11]. The methods used in the model to distill BERT while retaining good metrics were not used as we were unable to change the architecture of the model in our notebook and doing so would have required a significant amount of time.

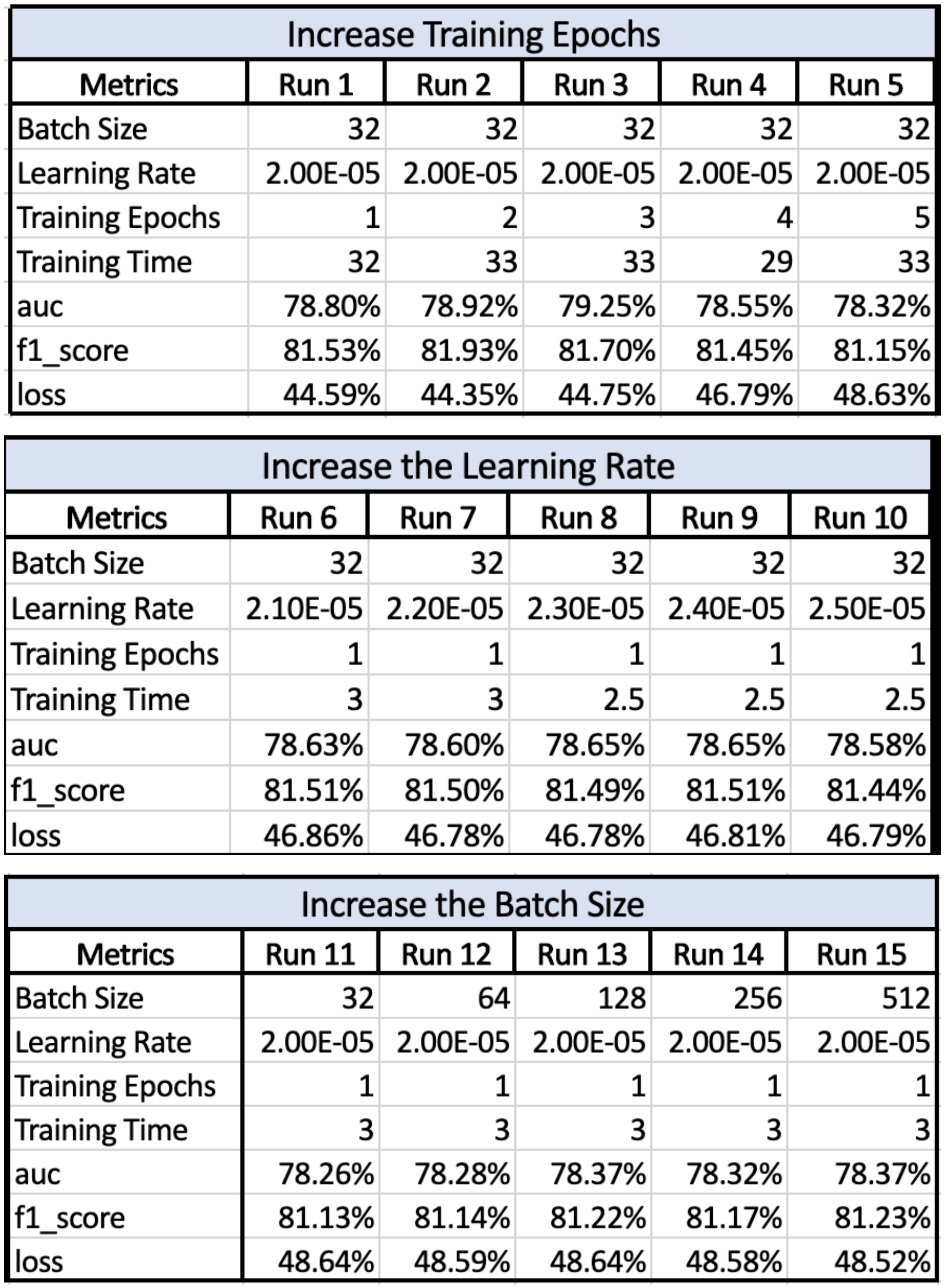
Other

There were other things that could be done to speed up BERT many they would require time and skill that the team did not have the current time. Some of them can be found on the following citation [12].

Results

MobileBERT

Our results were inclusive in showing that any of our hyper parameter tweaks actually lead to an improvement in accuracy. As seen in Figure 5, we held 3 experiments, each with 5 runs. We increased training epochs, increased the learning rate using 1 epoch, and finally increased the batch size with 1 epoch. The F1 score did not change any significant amount for any changes made to the model.



Increase Training Epochs

Metrics	Run 1	Run 2	Run 3	Run 4	Run 5
Batch Size	32	32	32	32	32
Learning Rate	2.00E-05	2.00E-05	2.00E-05	2.00E-05	2.00E-05
Training Epochs	1	2	3	4	5
Training Time	32	33	33	29	33
auc	78.80%	78.92%	79.25%	78.55%	78.32%
f1_score	81.53%	81.93%	81.70%	81.45%	81.15%
loss	44.59%	44.35%	44.75%	46.79%	48.63%

Increase the Learning Rate

Metrics	Run 6	Run 7	Run 8	Run 9	Run 10
Batch Size	32	32	32	32	32
Learning Rate	2.10E-05	2.20E-05	2.30E-05	2.40E-05	2.50E-05
Training Epochs	1	1	1	1	1
Training Time	3	3	2.5	2.5	2.5
auc	78.63%	78.60%	78.65%	78.65%	78.58%
f1_score	81.51%	81.50%	81.49%	81.51%	81.44%
loss	46.86%	46.78%	46.78%	46.81%	46.79%

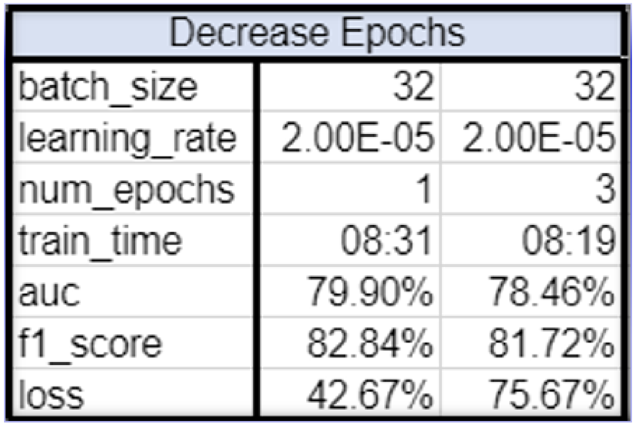
Increase the Batch Size

Metrics	Run 11	Run 12	Run 13	Run 14	Run 15
Batch Size	32	64	128	256	512
Learning Rate	2.00E-05	2.00E-05	2.00E-05	2.00E-05	2.00E-05
Training Epochs	1	1	1	1	1
Training Time	3	3	3	3	3
auc	78.26%	78.28%	78.37%	78.32%	78.37%
f1_score	81.13%	81.14%	81.22%	81.17%	81.23%
loss	48.64%	48.59%	48.64%	48.58%	48.52%

Figure 5: Metric results for MobileBERT. (top) varying training epochs, (middle) Varying learning rates with 1 epoch, (bottom) Varying batch sizes on 1 epoch

1 Epoch is Enough

When the epoch was reduced from three epochs to one epoch, the training data was also increased from threefold. There was a slight increase in training size and a small increase in the F1 score as Figure 6 shows. These results differ from the results in the paper that was referenced in the methods, this might have been due to not changing the iteration parameter and small dataset used.



Decrease Epochs		
batch_size	32	32
learning_rate	2.00E-05	2.00E-05
num_epochs	1	3
train_time	08:31	08:19
auc	79.90%	78.46%
f1_score	82.84%	81.72%
loss	42.67%	75.67%

Figure 6: Metric result for training with 1 epoch and an increased dataset

Conclusion

The parameters changes proposed in MobileBERT affected the BERT model in negligible ways while the “1 Epoch is Enough” method saw a slight increase in F1 score. The team would make several improvements to conduct a more precise analysis of hyper-parameter tuning with BERT. We would first spend additional time setting up a cloud environment that could process the BERT base model faster by finding the optimal batch size to use when training with TPU’s.

This paper has presented many methods that could be implemented to make BERT better, but many could not be utilized because of out time and ability constraints. To create a better hyper-parameter test we would need to look into changing more parameter in the source files of BERT and make some small architectural tweaks to the model.

References

[1] U. Malik, "Using Regex for Text Manipulation in Python," stackabuse, [Online]. Available: https://stackabuse.com/using-regex-for-text-manipulation-in-python/.

[2] G. Singh, "Why you should avoid removing STOPWORDS," towardsdatascience, 24 Jun 2019. [Online]. Available: https://towardsdatascience.com/why-you-should-avoid-removing-stopwords-aa7a353d2a52.

[3] arxiv.org, "AI_Project Predicting Movie Reviews with BERT on TF Hub.ipynb," colab.research.google.com, [Online]. Available: https://colab.research.google.com/drive/1pfuFzsyQwUeW9VEQt2y5UXW3Qs8_vCpX.

[4] R. Horev, "BERT Explained: State of the art language model for NLP," towardsdatascience, 10 Nov 2018. [Online]. Available: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.

[5] H. Y. X. S. R. L. Y. Y. D. Z. Zhiqing Sun, "MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices," arxiv.org, 2020.

[6] A. Komatsuzaki, "One Epoch Is All You Need," arxiv.org, 2019.

[7] "Sentiment Analysis on SST-2 Binary classification," paperswithcode, [Online]. Available: https://paperswithcode.com/sota/sentiment-analysis-on-sst-2-binary.

[8] O. L. G. N. Paul Michel, "Are Sixteen Heads Really Better than One?," in *33rd Conference on Neural Information Processing Systems*, Vancouver, Canada, 2019.

[9] J. L. S. R. J. H. S. K. S. B. X. S. J. D. K. K. C.-J. H. Yang YouYang You, "Large Batch Optimization for Deep Learning: Training BERT in 76 minutes," in *ICLR*, Berkeley, 2020.

[10] R. S. Nitish Shirish Keskar, "Improving Generalization Performance by Switching from Adam to SGD," arXiv, 20 Dec 2017. [Online]. Available: https://arxiv.org/abs/1712.07628v1.

[11] L. D. J. C. T. W. Victor Sanh, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," arXiv.org, 2 Oct 2019. [Online]. Available: https://arxiv.org/abs/1910.01108.

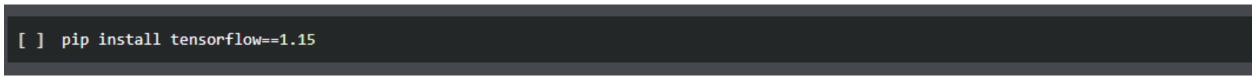
[12] G. Sapunov, "Speeding up BERT," blog.inten.to, 30 Sept 2019. [Online]. Available: https://blog.inten.to/speeding-up-bert-5528e18bb4ea.

User Manual for BERT using Google Colab

The Google Colab notebook already has text and instruction, but a few important steps will be shown here. The original notebook is shown in [3] so only the parts that are changed or are important will be displayed as steps as once certain things are entered, most of the remaining steps can be automatically run.

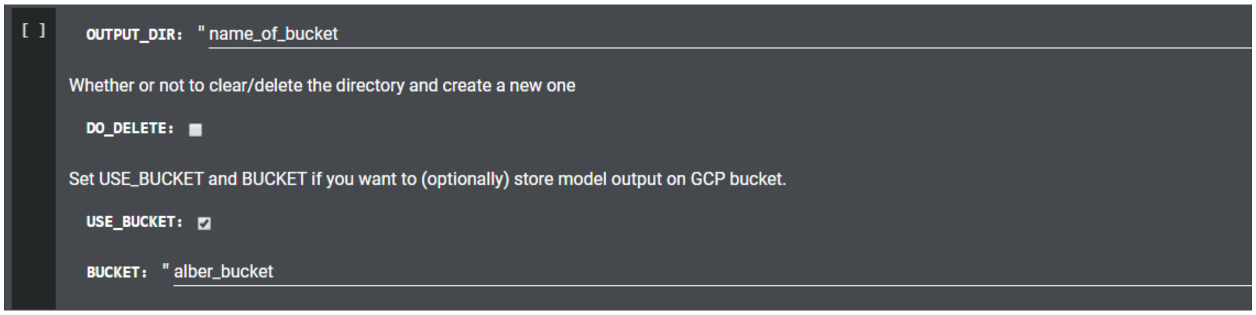
Environment Setup

BERT was originally trained under TensorFlow 1.11.0, it still works under many versions but 1.15m was chosen as that was the moist recent TensorFlow version in anaconda before TensorFlow 2.0.0. With the newer TF v2 version, many aspects of BERT have to be changed as some methods were removed and changed, look the TensorFlow 2 documentation for more information.



```
[ ]  pip install tensorflow==1.15
```

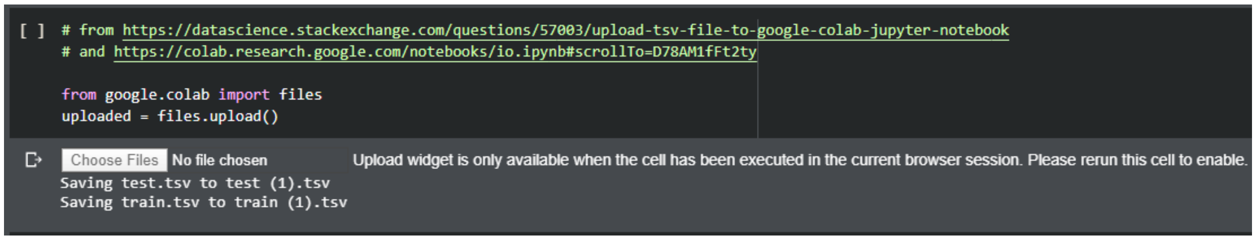
The notebook has a cell that allows you to connect to a bucket that is found in Google's cloud service, this will output all model checkpoints and evaluation directly to the chosen bucket. The OUTPUT_DIR is the name of the folder that will be created to contain all the outputs, and the BUCKET: option lets you pick what bucket you would like to use. The first time the cell is run a short verification is performed to make sure that the right person is trying to access the given bucket.



```
[ ]   OUTPUT_DIR:  " name_of_bucket

   Whether or not to clear/delete the directory and create a new one

      DO_DELETE: ■

   Set USE_BUCKET and BUCKET if you want to (optionally) store model output on GCP bucket.

      USE_BUCKET: ☑

      BUCKET:  " alber_bucket
```

Data input

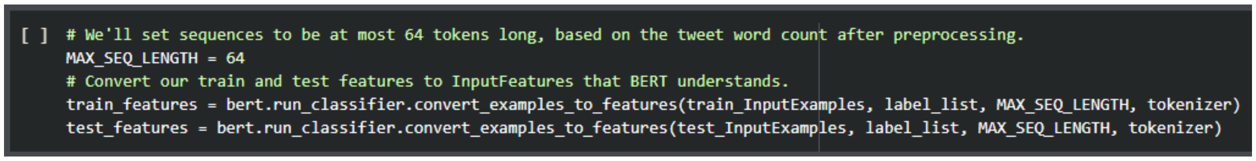
This cell lets you import and locally upload your data into the Google Colab notebook. Once its been uploaded then it can be accessed locally. You can also just find and drag files in the appropriate places, but this built cell does the work for you. The first time the cell is run a short verification is performed to make sure that the right person is trying to access the appropriate drive.



```
[ ] # from https://datascience.stackexchange.com/questions/57003/upload-tsv-file-to-google-colab-jupyter-notebook
    # and https://colab.research.google.com/notebooks/io.ipynb#scrollTo=D78AM1fFt2ty

    from google.colab import files
    uploaded = files.upload()

    Choose Files  No file chosen      Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
    Saving test.tsv to test (1).tsv
    Saving train.tsv to train (1).tsv
```

Sequence Length

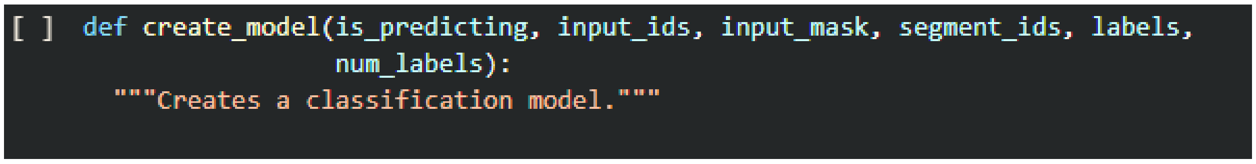
The sequence length can be changed, the default was 128 but it was changed to 64 after preprocessing the twitter data. Generally, the longer the sequence length, the more computationally expensive the model will be to train.



```
[ ] # We'll set sequences to be at most 64 tokens long, based on the tweet word count after preprocessing.
    MAX_SEQ_LENGTH = 64
    # Convert our train and test features to InputFeatures that BERT understands.
    train_features = bert.run_classifier.convert_examples_to_features(train_InputExamples, label_list, MAX_SEQ_LENGTH, tokenizer)
    test_features = bert.run_classifier.convert_examples_to_features(test_InputExamples, label_list, MAX_SEQ_LENGTH, tokenizer)
```

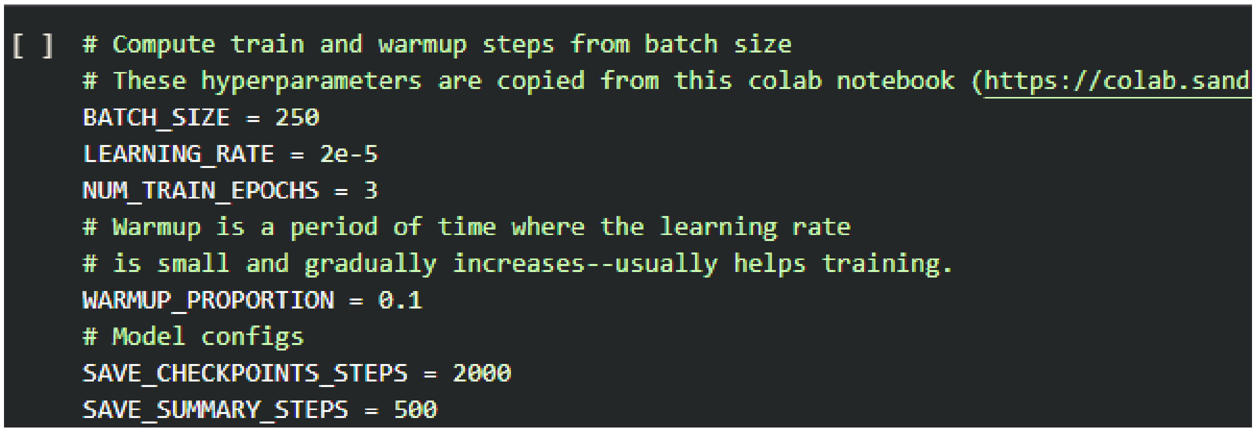
Creating the Model

The FC architecture of the model can be changed in this cell, but the guidelines to do so can be found in BERT's original source files.



```
[ ] def create_model(is_predicting, input_ids, input_mask, segment_ids, labels,
                     num_labels):
      """Creates a classification model."""
```

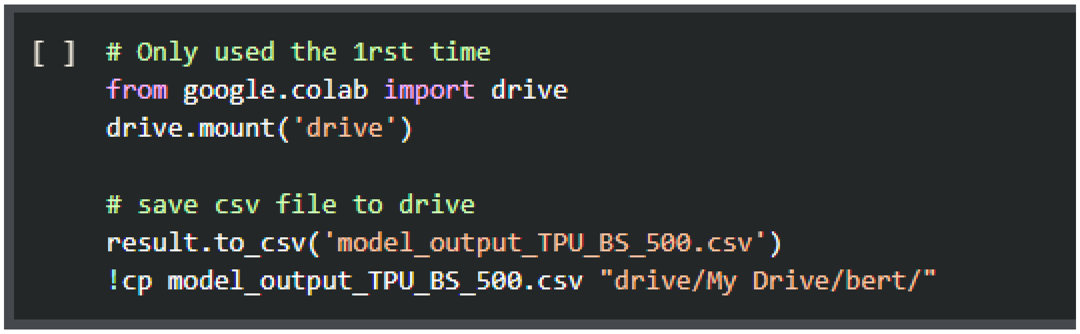
The major hyper-parameters can be changed in this cell. There is a limit on the batch size as you get a limited amount of memory allotted in Google Colab so if this number is set too high the model will terminate training and notify you. The learning rate and epoch number shown were used as the reference numbers. Warm-up describes how fast the learning rate will start up, it makes the learning rate small at first and the increases it over a few iterations or epochs until it reaches the actual learning rate.



```
[ ]  # Compute train and warmup steps from batch size
     # These hyperparameters are copied from this colab notebook (https://colab.sand
     BATCH_SIZE = 250
     LEARNING_RATE = 2e-5
     NUM_TRAIN_EPOCHS = 3
     # Warmup is a period of time where the learning rate
     # is small and gradually increases--usually helps training.
     WARMUP_PROPORTION = 0.1
     # Model configs
     SAVE_CHECKPOINTS_STEPS = 2000
     SAVE_SUMMARY_STEPS = 500
```

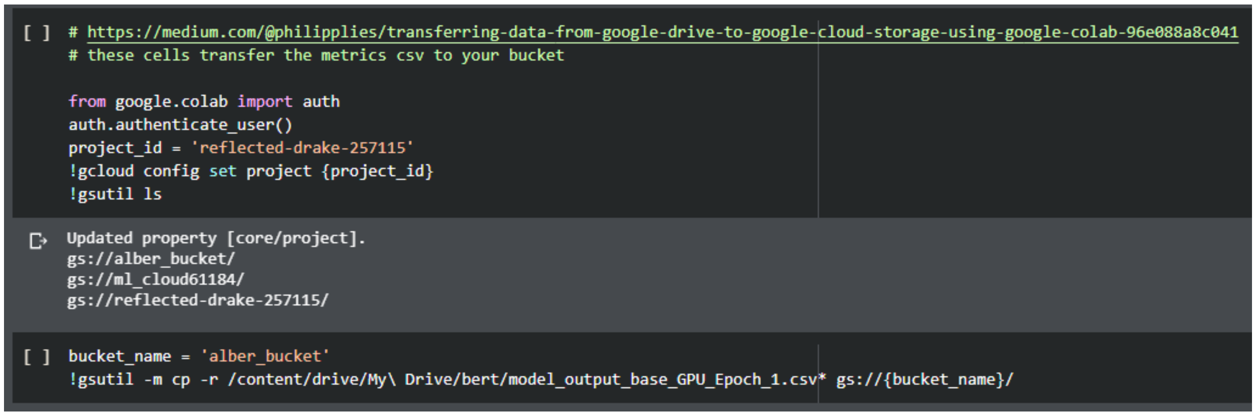
Model Evaluation

After the model has been trained and evaluated, a metrics csv file is generated containing a list of the hyper parameters and many of the metrics found in the model. There are two places that the file can be put, your local cloud drive or on your bucket on the folder that was specified at the beginning. The cell below gives the user the ability to save the file their chosen drive directory with using their chosen file name.



```
[ ]  # Only used the 1rst time
     from google.colab import drive
     drive.mount('drive')

     # save csv file to drive
     result.to_csv('model_output_TPU_BS_500.csv')
     !cp model_output_TPU_BS_500.csv "drive/My Drive/bert/"
```

The cells used to save the file on the bucket is shown below. The gsutil command lets you access your cloud storage and copies the csv file from your drive to your bucket. Your particular Project ID should be entered in the “project_id” line, ad your bucket name has to be entered in the bucket_name line.



```
[ ]  # https://medium.com/@philipplies/transferring-data-from-google-drive-to-google-cloud-storage-using-google-colab-96e088a8c041
     # these cells transfer the metrics csv to your bucket

     from google.colab import auth
     auth.authenticate_user()
     project_id = 'reflected-drake-257115'
     !gcloud config set project {project_id}
     !gsutil ls

     Updated property [core/project].
     gs://alber_bucket/
     gs://ml_cloud61184/
     gs://reflected-drake-257115/

[ ]  bucket_name = 'alber_bucket'
     !gsutil -m cp -r /content/drive/My\ Drive/bert/model_output_base_GPU_Epoch_1.csv* gs://{bucket_name}/
```